Combinatorial Mathematics

Mong-Jen Kao (高孟駿)

Monday 18:30 – 20:20

Outline

- The Maximum Matching Problem
  - A Generic Algorithm and the Berge’s Theorem
  - Solving the Augmenting Path Problem
    - DFS-based & BFS-based Algorithms for Bipartite Graphs
    - The Blossom Algorithm for General Graphs
- Concluding Notes
  - The best algorithms for Maximum Matching

The Maximum Matching Problem

- The Berge's theorem suggests the following simple algorithm.
  - Let $G = (V, E)$ be the input graph.

1. $M \longleftarrow \emptyset$.
2. Repeat until there is no M-augmenting path in G.
   a. Compute an M-augmenting path P.
   b. Set $M \longleftarrow (M \setminus P) \cup (P \setminus M)$.
3. Output M.

The Augmenting Path Problem

- To solve the maximum matching problem,
  it suffices to answer the *augmenting path problem*.

- Input :
  - A graph $G = (V, E)$ and a matching M for G.

- Goal :
  - Compute an M-augmenting path for G, or,
    Assert that there exists no such path.

The Augmenting Path Problem

- To solve the maximum matching problem,
  it suffices to answer the following *augmenting path problem*.

- In this lecture, we will introduce algorithms that solve the augmenting path problem.
  - $O(m)$ for bipartite graph.
  - $O(nm)$ for general graphs.

The Augmenting Path Problem in Bipartite Graphs

For bipartite graphs,
the augmenting path problem can be solved by simple DFS in $O(n+m)$ time.

The Augmenting Path Problem in Bipartite Graphs

- Let $G = (V, E)$ be a bipartite graph *with partite sets* $\boldsymbol{A}$ ***and*** $\boldsymbol{B}$, and $\boldsymbol{M}$ be a matching for G.

- We introduce an algorithm that computes in $O(m)$ time either
  - An M-augmenting path for G, or,
  - A vertex cover C for G with $|C| = |M|$.

  Note that, in the latter case, M is a maximum matching by the weak duality, and hence no augmenting path exists.

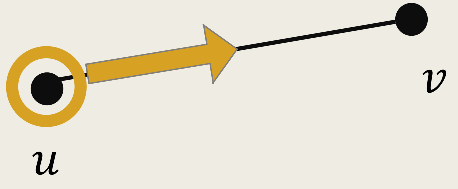
An Augmenting Path Algorithm for Bipartite Graphs

- Let $G = (V, E)$ be a bipartite graph *with partite sets* $\boldsymbol{A}$ ***and*** $\boldsymbol{B}$, and $\boldsymbol{M}$ be a matching for G.

- The algorithm attempts to compute an M-augmenting path ***<u>starting at an unmatched vertex in</u>*** $\underline{A}$ using a DFS-based recursive procedure ***aug-path()***.
  - If it succeeds for some unmatched vertex $v \in \boldsymbol{A}$, then we're done.
  - If it fails for *every unmatched vertex* in $\boldsymbol{A}$, then a vertex cover C with $|C| = |M|$ can be defined.

The DFS-based Recursive Procedure ***aug-path()***

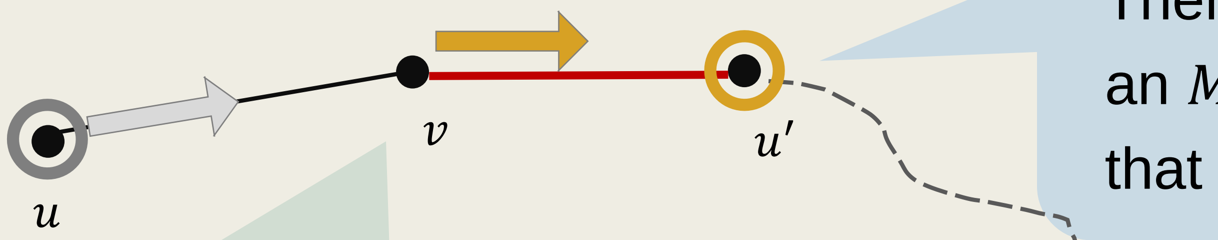
- Finding an augmenting path in a bipartite graph
  can be handled by a simple & intuitive DFS-based procedure.
  - We start with an unmatched vertex, say, u.
    - The goal is to find an M-augmenting path starting from u.
  - Consider ***each neighbor*** *of* u, say, v.



If v is *<u>unmatched</u>*,
then u, v is an M-augmenting path,
and we're done.

- We start with an unmatched vertex, say, u.
  - Our goal is to find an M-augmenting path starting from u.

- Consider each neighbor of u, say, v.



Then, the goal becomes finding an M-augmenting path that starts that ***starts from*** u'.

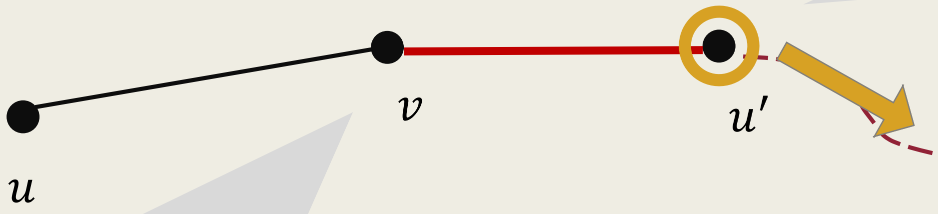
If v is ***matched***, then
to form an M-augmenting path that passes v,
we must follow the matched edge to some u'.

This is a recursive problem that starts at the vertex u'.

– We start with an unmatched vertex, say, u.

  - Our goal is to find an M-augmenting path starting from u.

– Consider each neighbor of u, say, v.



Then, the goal becomes finding an M-augmenting path that starts that ***starts from*** u'.

This is a recursive problem that starts at the vertex u'.

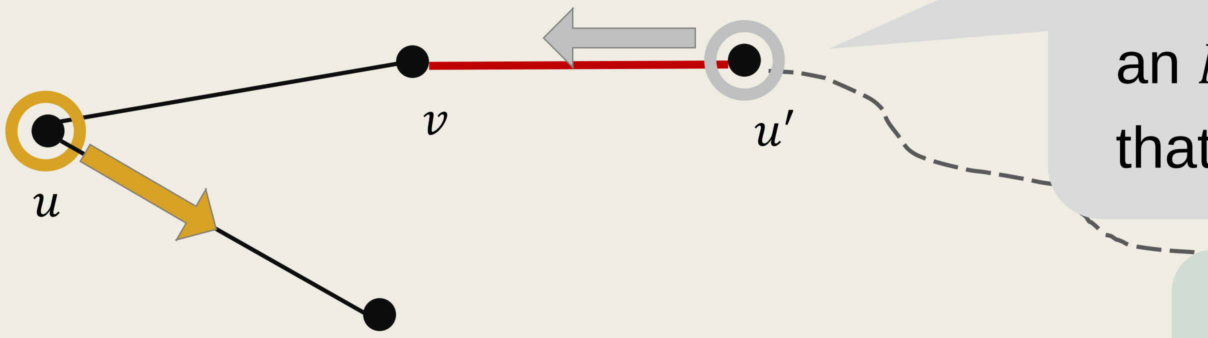
If v is ***matched***, then
to form an M-augmenting path that passes v,
we must follow the matched edge to some u'.

If the recursion succeeds,
we have an augmenting path for u.

– We start with an unmatched vertex, say, u.

  - Our goal is to find an M-augmenting path starting from u.

– Consider each neighbor of u, say, v.



Then, the goal becomes finding an M-augmenting path that starts that ***starts from*** u'.

This is a recursive problem that starts at the vertex u'.

If it fails, then we go back to u,
and continue to examine the next neighbor
until all its neighbors have been examined.

The DFS-based Recursive Procedure ***aug-path()***

- To formally describe the procedure,
  let's assume the following.

  - Each vertex in G is associated with a status,
    which is either *visited* or *unvisited*.

  - For each vertex v,
    let $\mathrm{match}[v]$ denote the vertex to which v is matched.

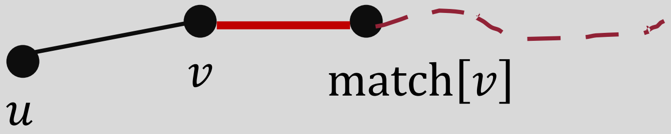
    - $\mathrm{match}[v] = -1$ if v is unmatched.

- The DFS-based recursive procedure goes as follows.

<u>Procedure</u> Aug-Path(u)

1. Mark u as *visited*.
2. For each neighbor v of u, do.
    - If v is *unmatched*, then return the path $\{u, v\}$.
    - If $\mathrm{match}[v]$ is *unvisited* and $(\, P \leftarrow \text{Aug-Path}(\mathrm{match}[v])\,) \neq \emptyset$, then return the path $\{u, v, P\}$.
3. Return $\emptyset$.

Augmenting path from $\mathrm{match}[v]$ is found.



An Augmenting Path Algorithm for Bipartite Graphs

- Let $G = (V, E)$ be the input bipartite graph with partite sets A and B, and M be a matching for G.

<u>An Augmenting Path Algorithm</u> (for Bipartite Graphs).

1. Mark all the vertices as *unvisited*.
2. For each *unmatched* vertex $u \in A$, do
   - If ($P \leftarrow$ Aug-Path(u)) $\neq \emptyset$, then return P.
3. Report "No" and return a <u>*vertex cover*</u> C with $|C| = |M|$.

We will show how this can be done.

Analysis of the Algorithm

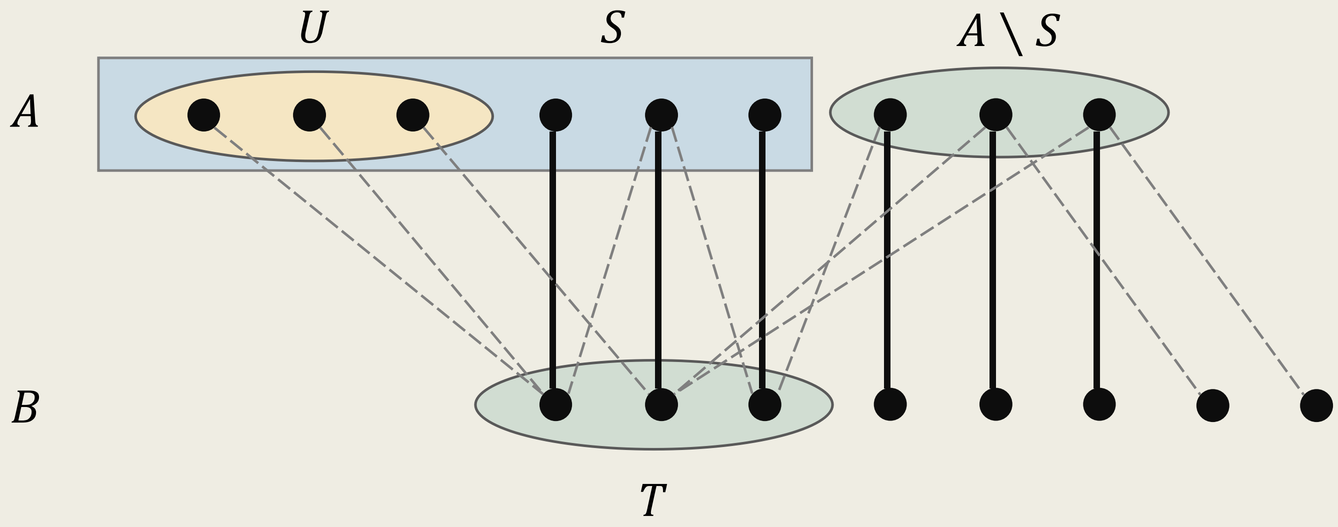
- Since each vertex is visited at most once and each edge is examined at most twice by the procedure Aug-Path(),
  - The algorithm runs in $O(n+m)$ time.

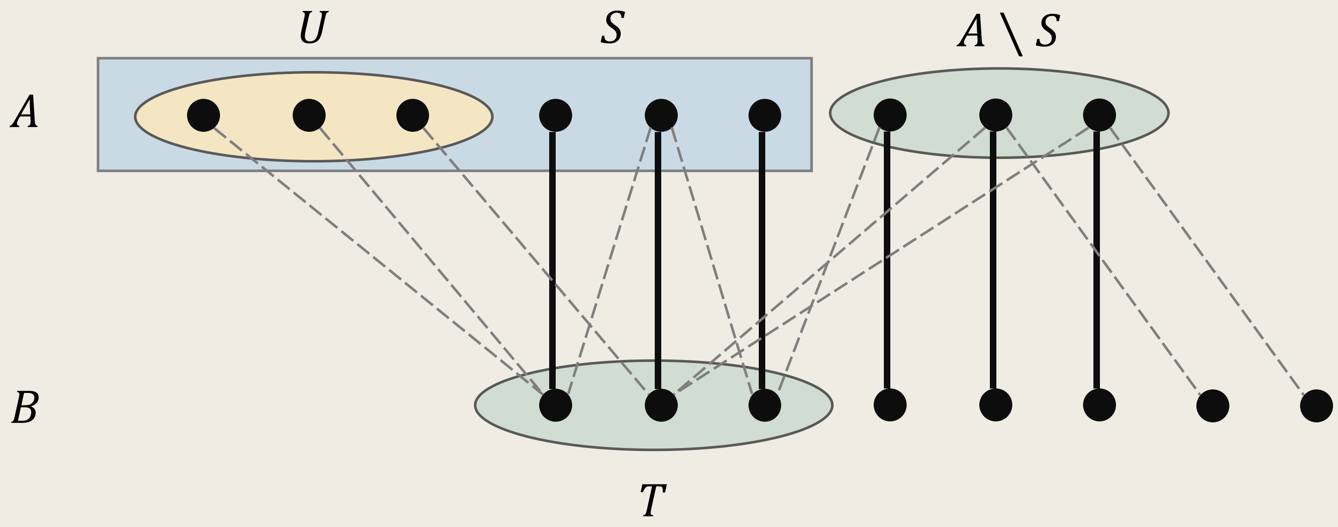
- It is clear that, if Aug-Path(u) returns a non-empty path P, then an M-augmenting path starting at u is found.

- To prove the correctness of the algorithm, we need to prove that,
  - There exists no M-augmenting path in the graph when the algorithm reports "No."

Notations

- Let A and B be the two partite sets of G.
  - Let U be the set of unmatched vertices in A.
  - Let S be the vertices in A that are marked as *visited*.
  - Let T be the set of vertices in B that are matched to $S \setminus U$ by M.





Theorem 3.

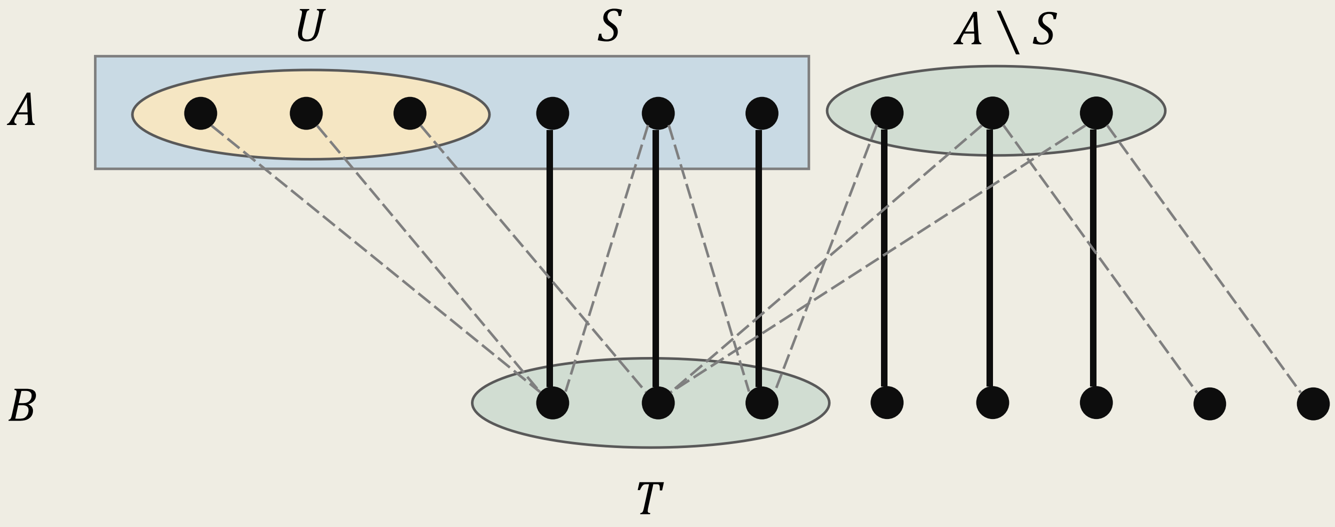
If the Augmenting Path Algorithm reports "No," then the set $C \coloneqq (A \setminus S) \cup T$ is a vertex cover for G with size M.

Note that, this is also a *constructive proof* for the König-Egeváry theorem.

Observation 1.

Since v is marked visited,
it is visited by a recursion call that
originates from some $u \in U$.

- For any $v \in S \setminus U$,
  - There is an M-alternating path that starts at some $u \in U$ and ends at v with a matched edge in M.

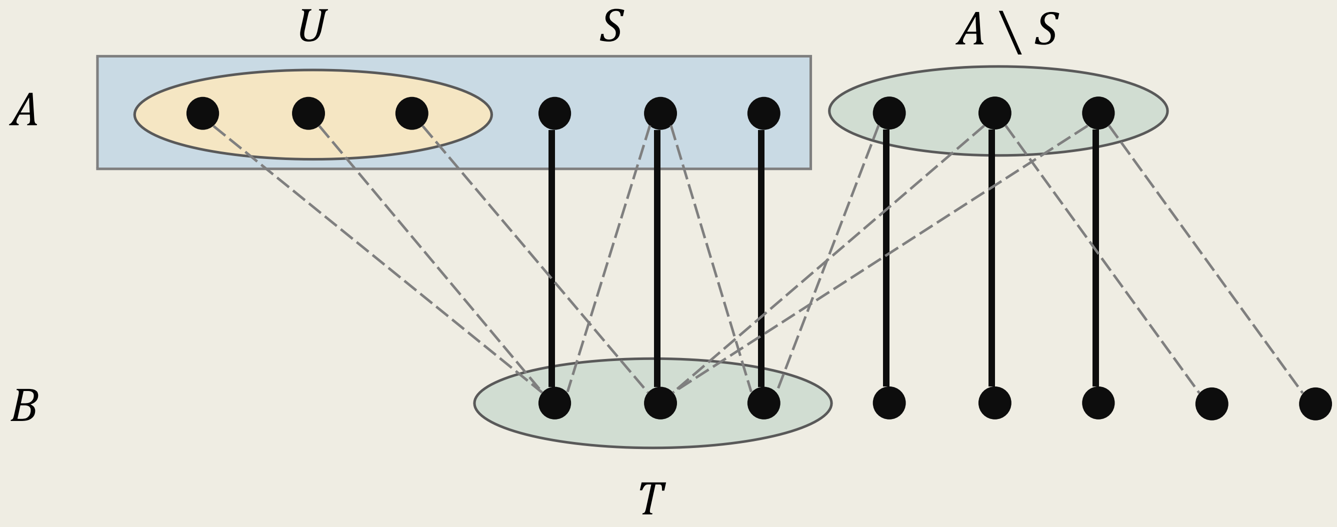


Observation 2.

- There exists no edge between S and $B \setminus T$.
  - Vertices in $A \setminus S$ are unvisited.

    Hence, there exists no edge between S and the matched vertices in $B \setminus T$.

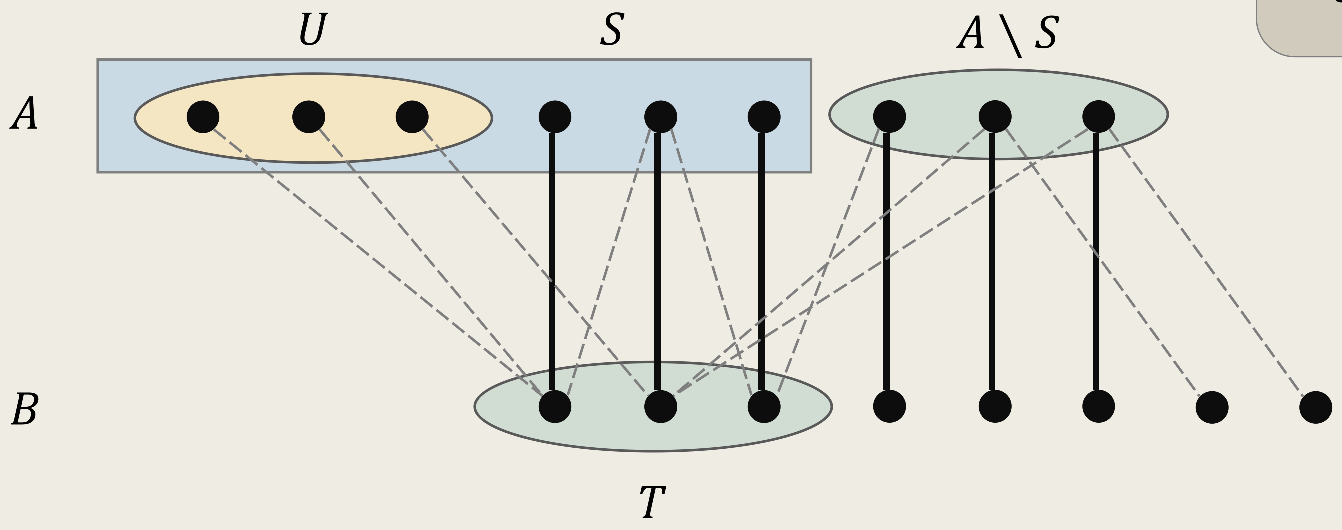
Otherwise, that matched vertex should be in T.



Observation 2.

- There exists no edge between S and $B \setminus T$.
  - If there exists an edge between S and some unmatched vertex in B, it will form an augmenting path that will be found by the recursive procedure.

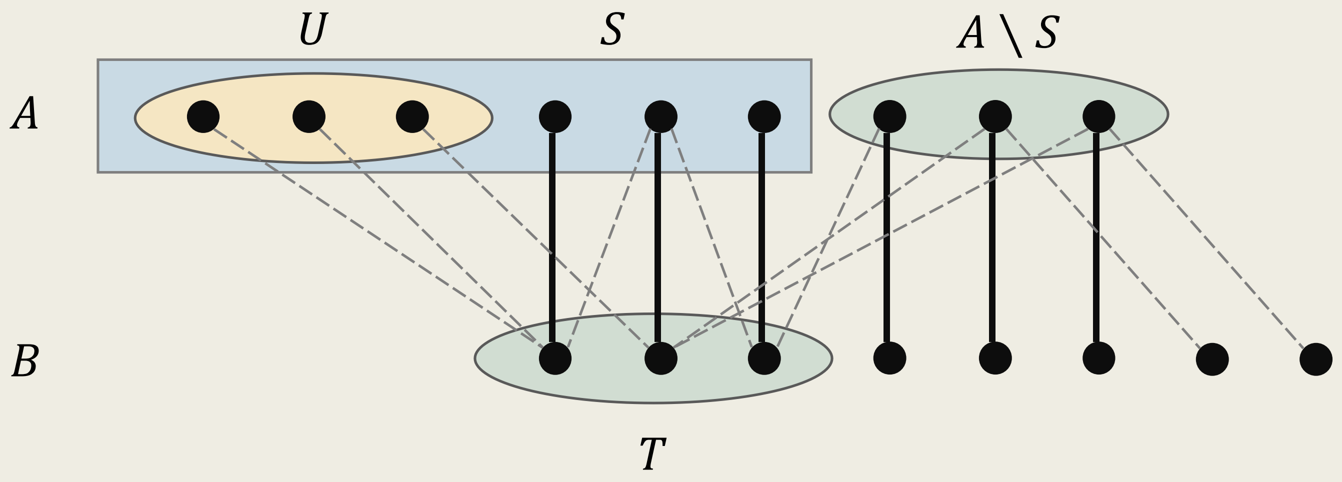
A contradiction since the algorithm reports "No."



Theorem 3.

If the Augmenting Path Algorithm reports "No," then the set $C \coloneqq (A \setminus S) \cup T$ is a vertex cover for G with size M.

- The edges between S and T can be covered by T.
- By Observation 2, the remaining edges can be covered by $A \setminus S$.
- Hence, C is a vertex cover for G.



Characterization of Bipartite Graphs

Identify the two partite sets of a bipartite graph when it is not given.

Characterization of Bipartite Graphs

- The following theorem is simple and intuitive to prove.

> **<u>Theorem. (Characterization of Bipartite Graphs)</u>**
>
> A graph $G = (V, E)$ is bipartite if and only if it has a 2-coloring, i.e., a 2-coloring for V such that no edge $e \in E$ is monochromatic.

- Note that, the 2-colorability of G can be tested by a simple DFS.
  - If G has a 2-coloring, then it also corresponds to a valid classification of the two partite sets.

You will need this fact in ProgHW #1.

An Alternative BFS-based Algorithm

An Alternative Algorithm

- Let X_0 be the set of all unmatched vertices in G.
- For any $i = 0, 1, 2, \ldots$, define
  - X_{2i+1} to be the set of *unvisited* vertices (not in $X_{\leq 2i}$) that can be reached from X_{2i} ***using an edge not in*** $\boldsymbol{M}$.
  - X_{2i+2} to be the set of *unvisited* vertices (not in $X_{\leq 2i+1}$) that can be reached from X_{2i+1} **using an edge in** $\boldsymbol{M}$.

An Alternative Algorithm

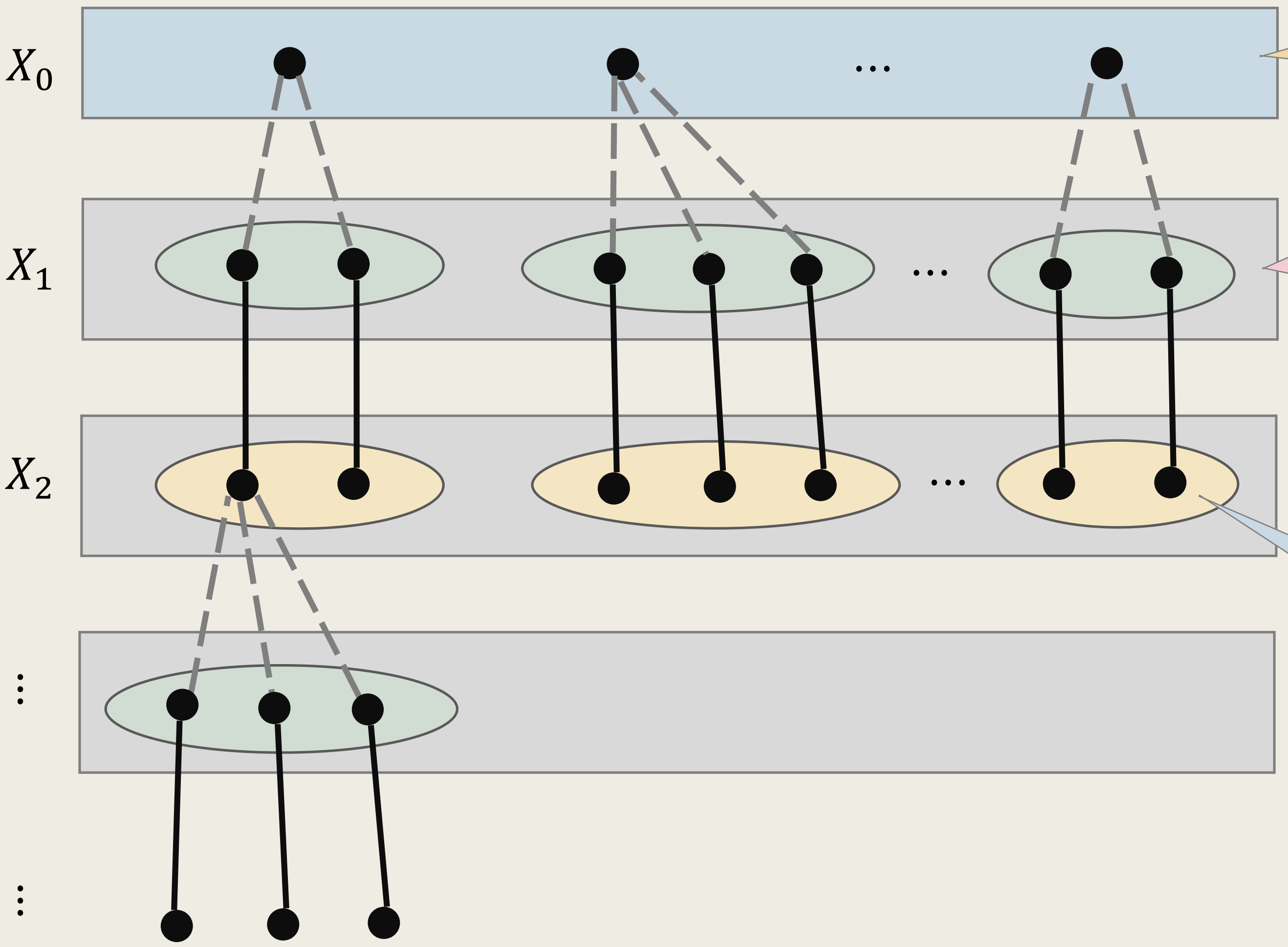
- Let X_0 be the set of all unmatched vertices in G.
- Formally, for any $i = 0, 1, 2, \ldots$, define

$$X_{2i+1} \coloneqq \{\ v \in V \setminus X_{\leq 2i}\ :\ \exists u \in X_{2i}\ s.t.\ (u, v) \notin M\ \}$$

and

$$X_{2i+2} \coloneqq \{\ v \in V \setminus X_{\leq 2i+1}\ :\ \exists u \in X_{2i+1}\ s.t.\ (u, v) \in M\ \}.$$

The ***Alternating Forest*** Formed by X_i



The set of unmatched vertices

The vertices in the ***odd levels*** have ***exactly one children*** in the forest.

Each non-root vertex has ***exactly one parent*** in the forest.

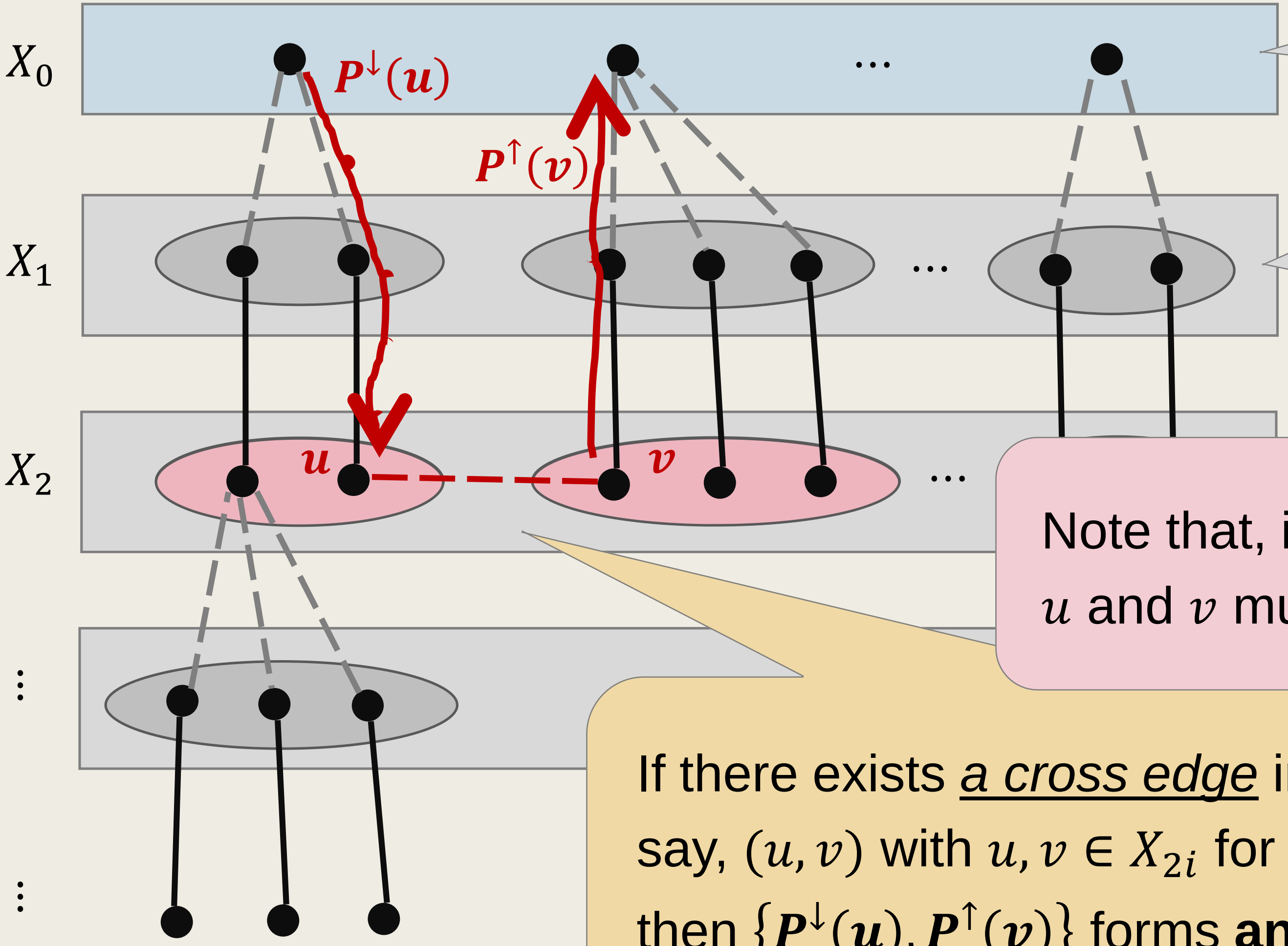
The ***Alternating Forest*** Formed by X_i

- The roots are the unmatched vertices in X_0.
  - Each non-root vertex has exactly one parent in the forest.

- For any vertex $v \in V$,
  - Let $P^{\uparrow}(v)$ be the path from v to its root in the forest.
  - Also, let $P^{\downarrow}(v)$ be the path from its root to v in the forest.

- Note that, $P^{\uparrow}(v)$ and $P^{\downarrow}(v)$ are uniquely defined, and they are M-alternating paths.

The ***Alternating Forest*** Formed by X_i



The set of unmatched vertices

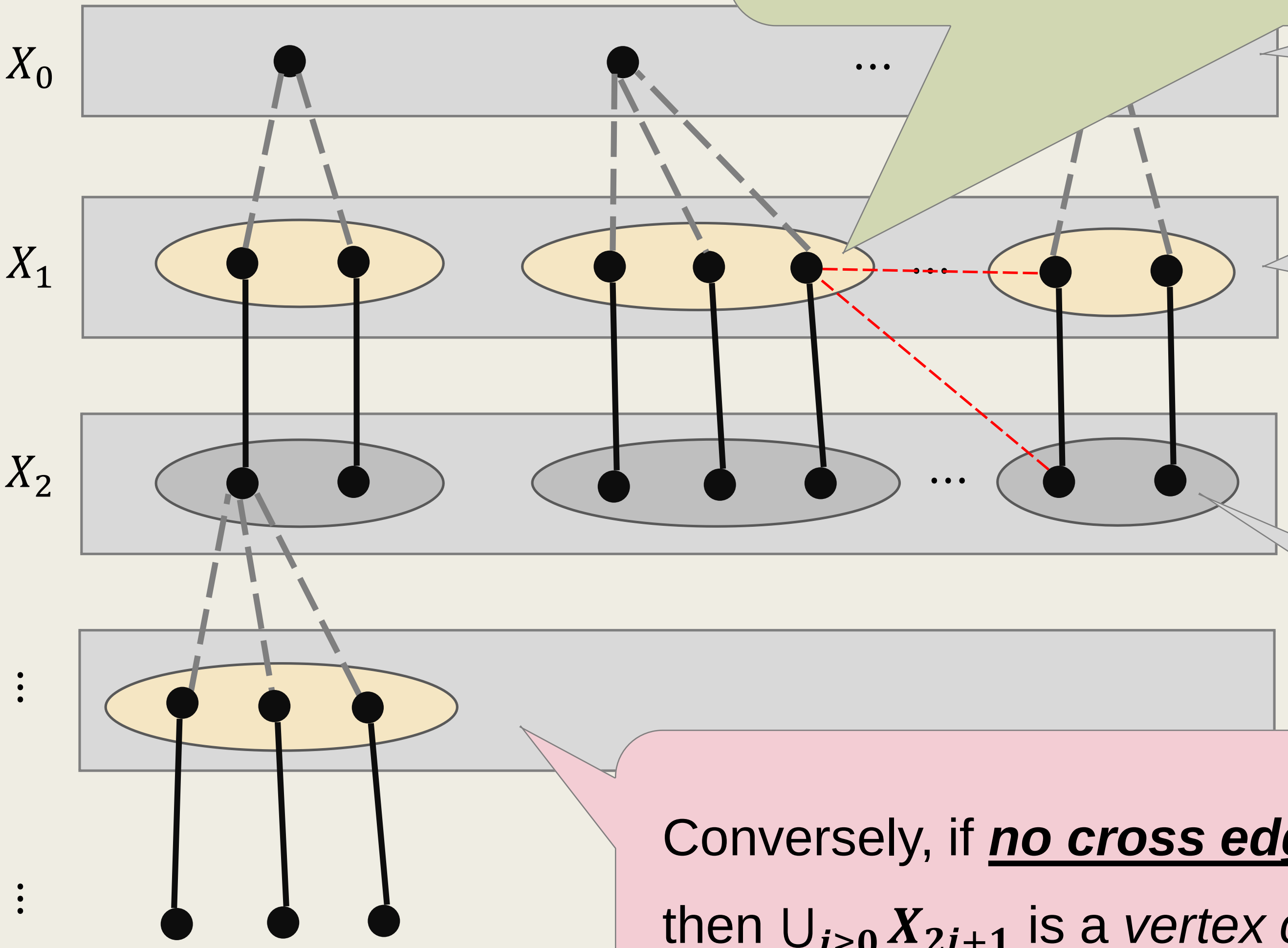
The vertices in the ***odd levels*** have ***exactly one children*** in the forest.

Note that, in this case,
u and v must be in different trees. (Why?)

If there exists <u>*a cross edge*</u> in <u>*the even levels*</u>,
say, (u, v) with $u, v \in X_{2i}$ for some i,
then $\{\boldsymbol{P^{\downarrow}(u)}, \boldsymbol{P^{\uparrow}(v)}\}$ forms **an $\boldsymbol{M}$-augmenting path**!

The ***Alternating Fore***

Note that, there may still be edges between a vertex in the odd level and other vertices, but we don’t care.



The vertices in the ***odd levels*** have ***exactly one children*** in the forest.

Conversely, if ***no cross edge*** exists ***in the even levels***, then $\bigcup_{i\geq 0} \boldsymbol{X_{2i+1}}$ is a *vertex cover* with size $|M|$!

- Let $G = (V, E)$ be a bipartite graph and M be a matching for G.

<u>Another BFS-based Augmenting Path Algorithm</u> (for Bipartite Graphs).

1. Let X_0 be the set of unmatched vertices and $t \leftarrow 0$.
2. Repeat until $X_{\leq 2t} = V$, do
   - If there exists an edge $(u, v) \in E$ for some $u, v \in X_{2t}$,
     then return the path $\{P^{\downarrow}(u), P^{\uparrow}(v)\}$.
   - Otherwise,
     form X_{2t+1} and X_{2t+2} as described and set $t \leftarrow t + 1$.
3. Report $\bigcup_{i \geq 0} X_{2i+1}$ as a *vertex cover* with size $|M|$.

The Augmenting Path Problem

in General Graphs

For general graphs,
the augmenting path problem can be solved in $O(nm)$ time *via proper vertex contractions*.

The Augmenting Path Problem in General Graphs

- Let $G = (V, E)$ be a general graph and $\boldsymbol{M}$ be a matching for G.

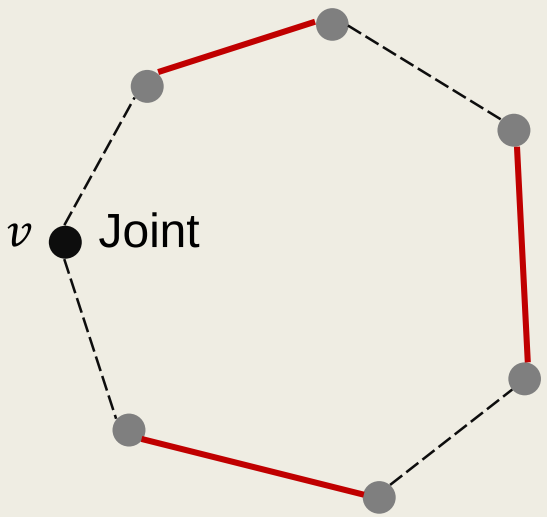
- We introduce an algorithm that computes in $O(nm)$ time either
  - An M-augmenting path for G, or,
  - A *structure* ***(proof)*** *showing that* ***M is maximum***.

    Hence, no M-augmenting path exists in the graph.

Note that, we can *no longer* count on *vertex covers* for this, since the ***strong duality does not hold*** between *matchings and vertex covers* in *general graphs*.

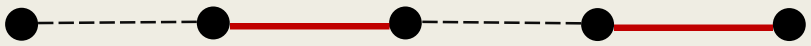
Blossom, Stem, and Flowers

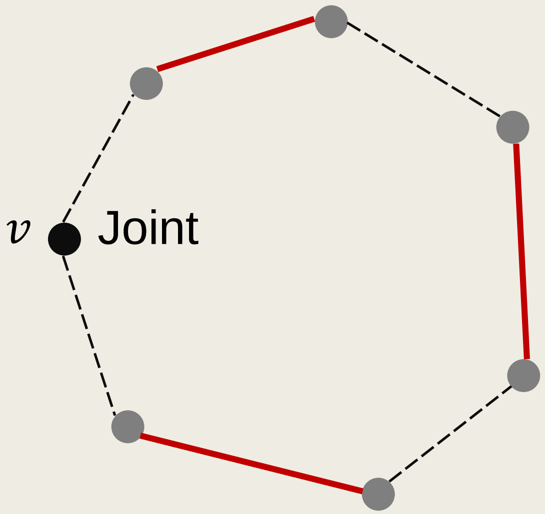
- A ***<u>blossom</u>*** is a cycle C
  with *<u>an odd length</u>* and $\lfloor |C|/2 \rfloor$ *<u>matched edges</u> in* M.
  - The vertex $v \in C$ that is not incident to any matched edge is called the "***<u>joint</u>***" of the blossom.



Blossom, Stem, and Flowers

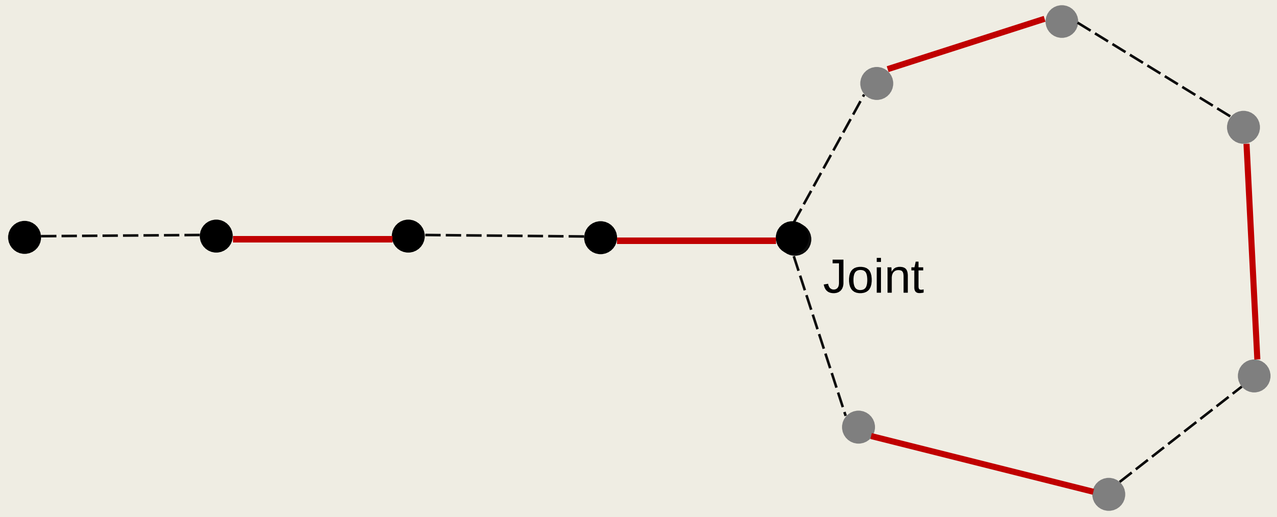
- A ***stem*** is an M-alternating path
  with *an even length* and *ends at a matched edge in* M.





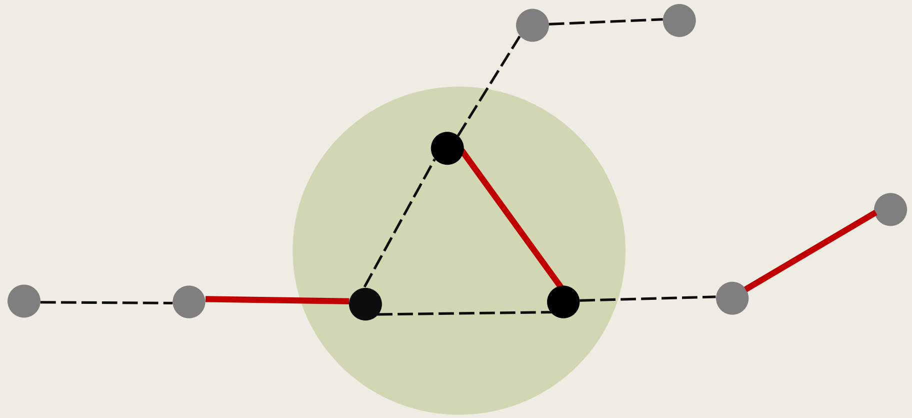
Blossom, Stem, and Flowers

- A ***flower*** is *a stem* and *a blossom* such that the stem ends at the joint of the blossom.

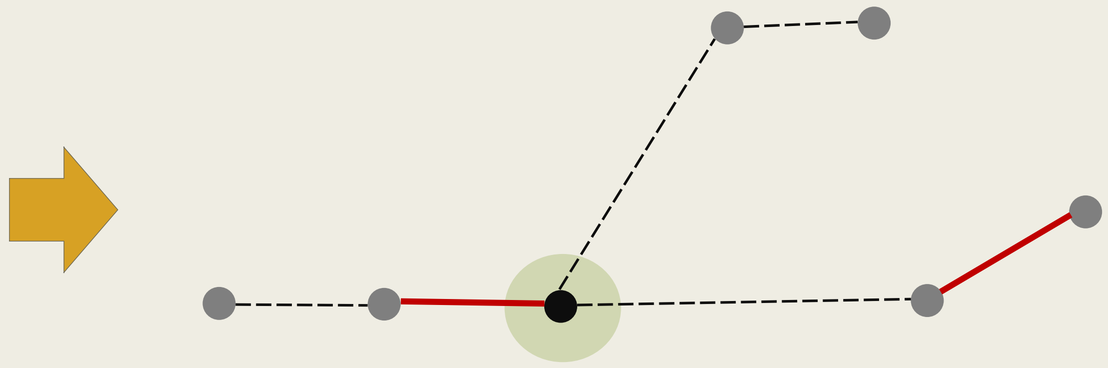


Contracting a Blossom

- Let C be a blossom in G.
  - Define G_C to be the graph obtained by contracting C in G, and M'_C be the remaining set of matched edges.



The blossom C

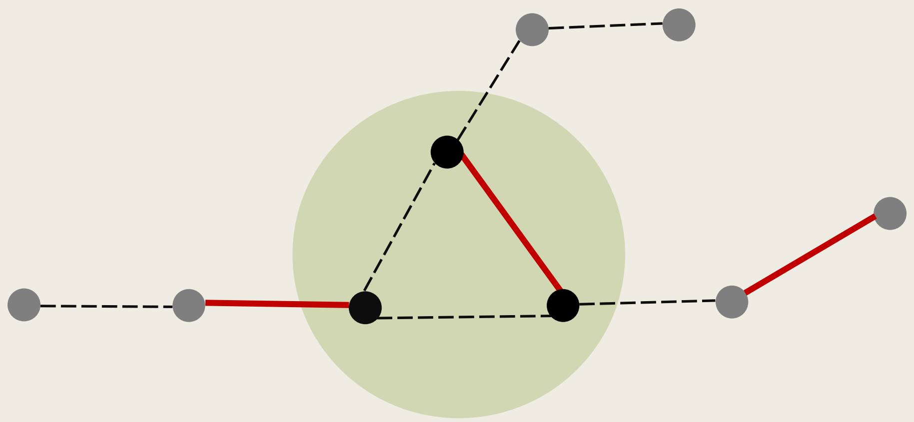


The contracted graph G_C

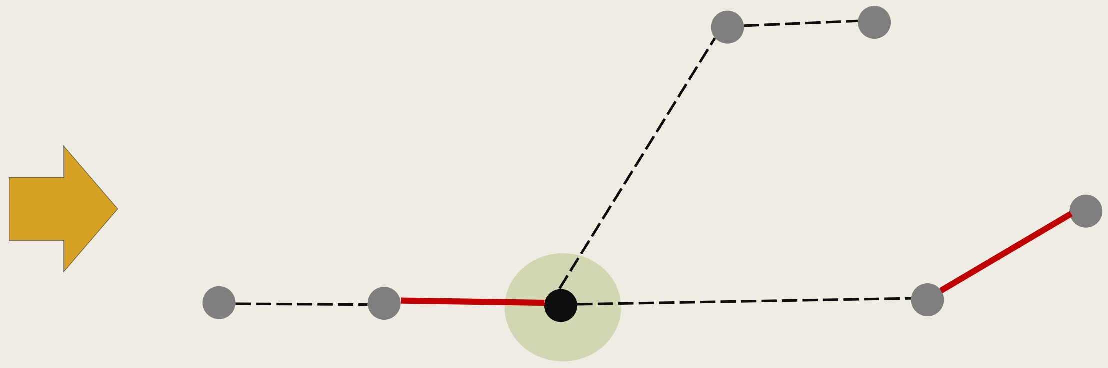
- Let C be a blossom in G.
  - Define G_C to be the graph obtained by contracting C in G, and M'_C be the remaining set of matched edges.

<u>Lemma. (Blossom Contraction)</u>

G has an M-augmenting path
if and only if G_C has an M'_C-augmenting path.



The blossom C



The contracted graph G_C

The Blossom Algorithm (by Jack Edmonds)

- Let X_0 be the set of all unmatched vertices in G.

- For any $i = 0, 1, 2, ...$, define

  - X_{2i+1} to be the set of *unvisited* vertices (not in $X_{\le 2i}$) that can be reached from X_{2i} ***using an edge not in $\boldsymbol{M}$***.

  - X_{2i+2} to be the set of *unvisited* vertices (not in $X_{\le 2i+1}$) that can be reached from X_{2i+1} **using an edge in $\boldsymbol{M}$**.

The Blossom Algorithm (by Jack Edmonds)

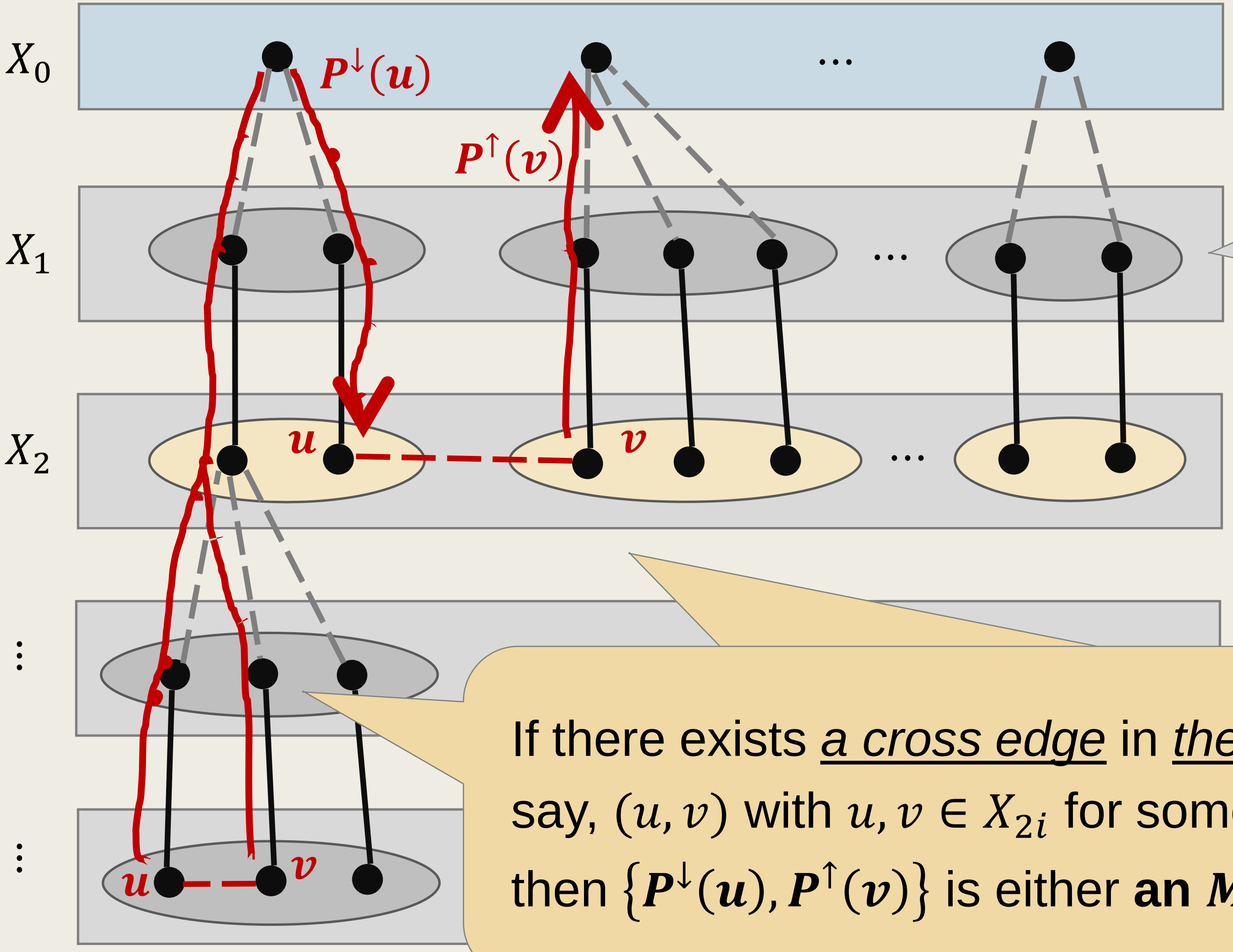
- Consider the alternating forest formed by X_i for all $i \geq 0$.
- If there exists ***a cross edge*** ***in an even level***,
  i.e., $(u, v) \in E$ for some $u, v \in X_{2i}$ and some $i \geq 0$,

  then $\{P^{\downarrow}(u), P^{\uparrow}(v)\}$ is either *an M-augmenting path* or *a flower*!

  – If $P^{\downarrow}(u) \cap P^{\uparrow}(v) = \emptyset$, then it is an augmenting path.

  – Otherwise,
  it is a flower with the common part being the stem.

The ***Alternating Forest*** Formed by X_i



The vertices in the ***odd levels*** have ***exactly one children*** in the forest.

If there exists *a cross edge* in *the even levels*, say, (u, v) with $u, v \in X_{2i}$ for some i, then $\{P^{\downarrow}(u), P^{\uparrow}(v)\}$ is either **an M-augmenting path or a flower**!

- Let $G = (V, E)$ be a graph and M be a matching for G.

<u>The Blossom Algorithm</u> (by Jack Edmonds).

1. Let X_0 be the set of unmatched vertices and $t \leftarrow 0$.
2. Repeat until $X_{\leq 2t} = V$, do
   - If there exists an edge $(u, v) \in E$ for some $u, v \in X_{2t}$,
     - If $P^{\downarrow}(u) \cap P^{\uparrow}(v) = \emptyset$, then return the path $\{P^{\downarrow}(u), P^{\uparrow}(v)\}$.
     - Otherwise, let $C \leftarrow P^{\downarrow}(u) \, \Delta \, P^{\uparrow}(v)$. Apply the algorithm recursively on G_C and M'_C. *<u>Expand the result</u>* and return it.
   - Otherwise,
     form X_{2t+1} and X_{2t+2} as described and set $t \leftarrow t + 1$.
3. Report $\bigcup_{i \geq 0} X_{2i+1}$ as a ***<u>proof</u>***.

The Correctness of

the Blossom Algorithm

Analysis of the Algorithm

- For the correctness of the algorithm,
  - It is clear that, when the blossom algorithm returns an M-augmenting path, it is indeed a valid one.

  - We need to show that, when the algorithm returns a proof (reports “No”), M is indeed a maximum matching.

    For this, we will use the Tutte-Berge Max-Min Theorem.

Lemma. (Tutte-Berge Max-Min Theorem)

Let $G = (V, E)$ be a graph,

$U \subseteq V$ be a vertex subset, and $M \subseteq E$ be a matching.

Then we always have

$$|M| \leq \frac{|V| + |U| - \mathrm{odd}(G \setminus U)}{2},$$

where $\mathrm{odd}(G \setminus U)$ is the number of components with an odd size in $G \setminus U$.

- Later we will show that,
  the inequality holds with equality for properly chosen M and U.

Let $G = (V, E)$ be a graph, $U \subseteq V$ be a vertex subset, and $M \subseteq E$ be a matching. Then we always have

$$|M| \leq \frac{|V| + |U| - \mathrm{odd}(G \setminus U)}{2},$$

where $\mathrm{odd}(G \setminus U)$ is the number of odd components in $G \setminus U$.

- Consider the components in U and $G \setminus U$.
  - Each vertex in U is incident with at most one edge in M.
  - For the remaining components K in $G \setminus U$, it contains at most $\left\lfloor \frac{|K|}{2} \right\rfloor$ edges in M.

Since all endpoints of the edges in M are distinct.

We always have $|M| \leq \big(|V| + |U| - \mathrm{odd}(G \setminus U)\big)/2$,

where $\mathrm{odd}(G \setminus U)$ is the number of odd components in $G \setminus U$.

- Consider the components in U and $G \setminus U$.
  - Each vertex in U is incident with at most one edge in M.
  - For the remaining components K in $G \setminus U$, it contains at most $\left\lfloor \frac{|K|}{2} \right\rfloor$ edges in M.

Since all endpoints of the edges in M are distinct.

- Hence,

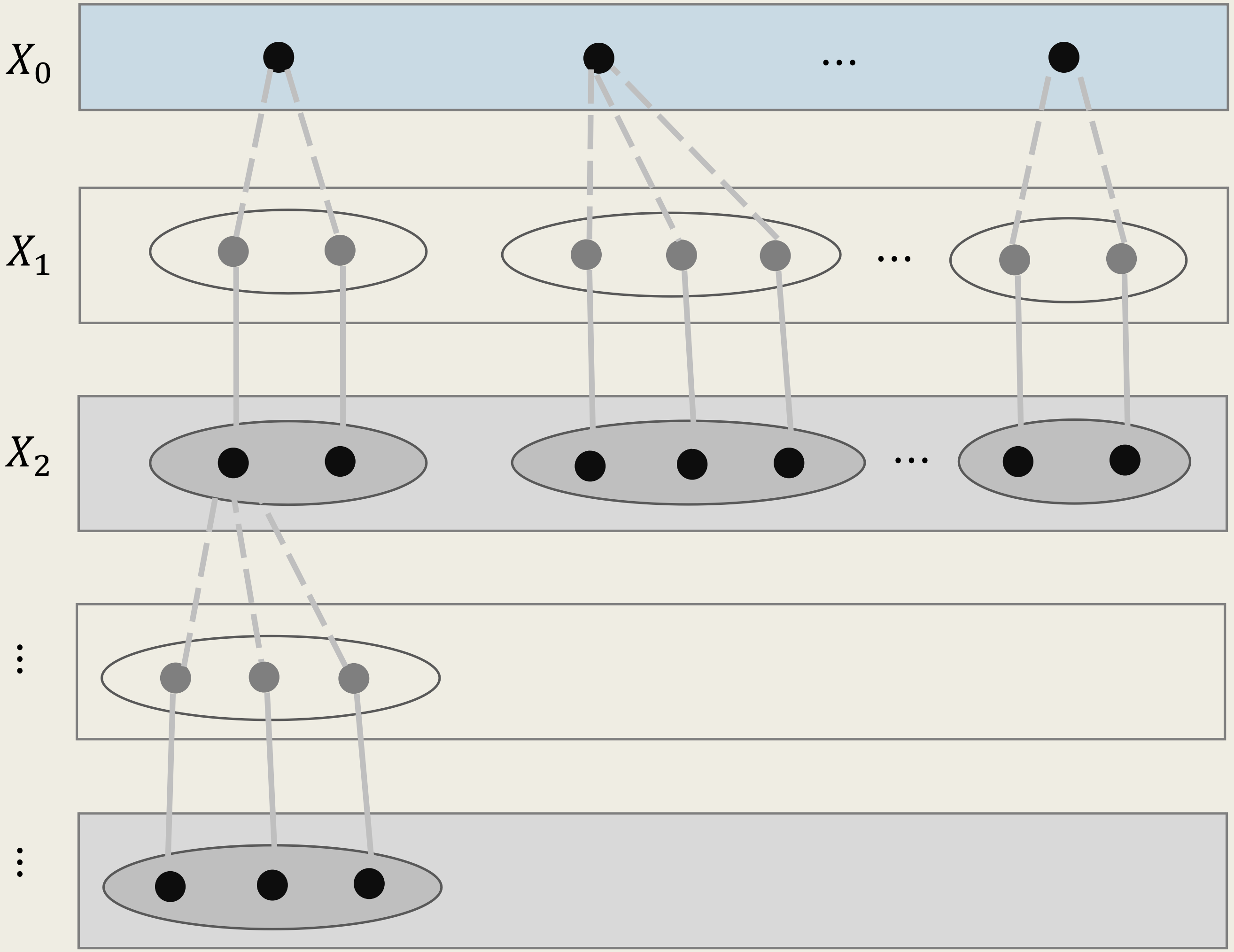
$$|M| \leq |U| + \sum_i \left\lfloor \frac{|K_i|}{2} \right\rfloor = |U| + \frac{|V| - |U|}{2} - \frac{\mathrm{odd}(G \setminus U)}{2}.$$

Analysis of the Algorithm

- Suppose that the Blossom algorithm returns a proof $\bigcup_{i\geq 0} X_{2i+1}$.
- By the Tutte-Berge's inequality, to prove that M is a maximum matching for G, it suffices to show that

The choice of $U \coloneqq \bigcup_{i\geq 0} X_{2i+1}$ will make
the Tutte-Berge's inequality hold with equality.

The ***Alternating Forest*** Formed by X_i



There is *no cross edge*
in the even levels.

Hence, the vertices in *the even levels* become **isolated**.

The *remaining components*
(not included in the forest)
are *perfectly matched* by M.

Analysis of the Algorithm

- Let $U \coloneqq \bigcup_{i\geq 0} X_{2i+1}$.
- Then,

$$|M| = |U| + \frac{|V \setminus X_{\geq 0}|}{2} = \frac{|V| + |U| - \left|\bigcup_{i\geq 0} X_{2i}\right|}{2}$$

$$= \frac{|V| + |U| - \mathrm{odd}(G \setminus U)}{2}.$$

- Hence, M is a maximum matching for G.

Concluding Notes

Best Algorithm for the Maximum Bipartite Matching

- In this lecture,
  we have seen an $O(nm) = O(n^3)$ algorithm for this problem.

- The best algorithm for this problem is the Hopcroft-Karp algorithm, which runs in $O\left(\sqrt{n}m\right) = O\left(n^{2.5}\right)$.

The Hopcroft-Karp Algorithm

- The idea is to perform a **BFS** simultaneously from all unmatched vertices in one partite set **to form alternating layers** until some unmatched vertices in the other partite set is met.
- Then a **layer-guided DFS** is used to construct a maximal set of <u>vertex-disjoint shortest augmenting paths</u>.
- It is guaranteed that, only $O(\sqrt{n})$ rounds are needed before the maximum matching is computed.

Best Algorithm for the Maximum Bipartite Matching

- This problem is a special case of the max-flow problem.

  A number of flow algorithms are applicable.

  - Practically,
    the most efficient one is the Dinic's algorithm.

  - Theoretically, the best algorithm is the "Almost linear-time" max-flow algorithm that runs in $m^{1+o(1)}$ time.

Maximum Matching in General Graphs

- For general graphs, we have seen the Edmonds Blossom Algorithm, which runs in $O(n^2 m) = O(n^4)$ time.

- The best (and more complicated) algorithm, due to Micali and Vazirani, solves this problem in $O(\sqrt{n}m) = O(n^{2.5})$ time.